# Formal Grammars for Linguistic Treebank Queries

**Mark Dras**
Centre for Language Technology
Macquarie University
madras@ics.mq.edu.au

**Steve Cassidy**
Centre for Language Technology
Macquarie University
Steve.Cassidy@mq.edu.au

## Abstract

There has been recent interest in looking at what is required for a tree query language for linguistic corpora. One approach is to start from existing formal machinery, such as tree grammars and automata, to see what kind of machine is an appropriate underlying one for the query language. The goal of the paper is then to examine what is an appropriate machine for a linguistic tree query language, with a view to future work defining a query language based on it. In this paper we review work relating XPath to regular tree grammars, and as the paper's first contribution show how regular tree grammars can also be a basis for extensions proposed for XPath for common linguistic corpus querying. As the paper's second contribution we demonstrate that, on the other hand, regular tree grammars cannot describe a number of structures of interest; we then show that, instead, a slightly more powerful machine is appropriate, and indicate how linguistic tree query languages might be augmented to include this extra power.

## 1 Introduction

There has been recent interest in looking at what is required for a query language for annotated linguistic corpora (Lai and Bird, 2004). These corpora are used in a range of areas of natural language processing (NLP)—parsing, machine translation, and so on—where they form the basis of training data for statistical methods; and also in linguistics, where they are used to extract examples of particular phenomena for analysis and testing of hypotheses. As noted by Lai and Bird, the prototypical hierarchical linguistic annotation is the syntax tree, and consequently the type of query language that is of interest is a tree query language.

One approach to deciding what is required in a tree query language, taken by Lai and Bird (2004), is to examine and compare a range of existing ones: for example, TGrep2 (Rohde, 2001), TIGERSearch (König and Lezius, 2001), or Emu (Cassidy and Harrington, 2001). One of their goals is to understand better the formal properties required of query languages.

It has been noted in a number of places that treebank querying is in essence a specification of a tree pattern, which matches against the desired trees in the treebank corpus (Abiteboul, 1997). These tree patterns can be described by existing formal machinery such as tree grammars and automata. An approach complementary to the one mentioned above is thus to examine the extent to which this formal machinery is adequate for describing tree patterns relevant for the sorts of queries of interest in NLP or linguistics, and then to link this to a query language. A reason for being interested in this link between tree query languages and formal machinery is that standard results are available for these latter. For example, an algorithm for recognition exists that is linear in the number of nodes in the tree and the size of the automaton; it is decidable whether the set of matches will be empty; and so on (Comon et al., 1997). Further, there is the promise of the availability of standard tools and efficient techniques that could be used by a tree query language. This is the case for formal machines over strings (for example, the library of finite-state string transducers of Mohri (1997)), although not yet for trees.

A number of researchers have looked at this complementary approach. One alternative is to design from scratch a tree query language derived from a tree grammar or automaton; this is taken by, for example, Chidlovskii (2000). Another is to relate an existing query language to a formal machine: Murata et al. (2000) present a taxonomy of XML schema languages using formal language theory.

In this paper, we follow the second of these alternatives. Existing work, mostly focussed on XML, has looked only at regular tree grammars and automata for modelling query languages; we examine the extent to which this is the case for linguistic treebanks, and what other machinery might be appropriate for a linguistic tree query language. We argue that while regular tree grammars might be satisfactory for a querying a broad range of phenomena, not all queries over trees representing natural language can be based on a regular tree grammar. This is a structural analogue of the work of Shieber (1985), which showed that natural language as a string language cannot be generated by a context-free grammar, which corresponds at the tree level to a regular tree grammar.

In Section 2 we give the definition of regular tree grammars, along with an approach used to relate them to XPath. In Section 3 we look at extensions to XPath that Cassidy (2003) argues are necessary for linguistic corpus querying, and show that these can be captured by regular tree grammars. In Section 4, however, we present some examples from Dutch from the work of Bresnan et al. (1982) to show that not all desired queries can be represented by regular tree grammars, and examine the question of what strong generative capacity is necessary in a tree grammar for representing natural language. Section 5 gives the definition of a more powerful grammar, the context-free tree grammar, and shows how this can describe queries related to the Dutch instance, along with what properties a query language based on these might have. Finally, Section 6 concludes.

## 2 Regular Tree Grammars

Regular tree grammars (RTGs) are in essence those trees whose paths are defined by regular grammars. Comon et al. (1997) provide an introduction to necessary concepts in tree grammars, along with well known results such as that the string languages yielded by RTG trees are the context-free languages. In their treatment they divide tree representations into two types: those for ranked trees (that is, where each symbol has a fixed number of children, with this number constituting the rank of the symbol), and those for unranked trees. XML schema languages typically use unranked trees, so we adopt, slightly modified, the definitions of these from Brüggemann-Klein et al. (2001) and Murata et al. (2000).

A regular tree grammar is a 4-tuple $G = (\Sigma, N, P, S)$ such that

- $\Sigma$ is a finite set of terminal symbols;
- $N$ is a finite set of nonterminal symbols;
- P is a finite set of productions of the form $X \rightarrow a(R)$, where $X \in N$, $a \in \Sigma$, and $R$ is a regular expression over $N \cup \Sigma$; and
- $S$ is the start symbol, $S \in N$.

Derivation $\Rightarrow$ (with transitive closure $\overset{*}{\Rightarrow}$) is defined in the usual way, with nonterminals symbols rewritten by means of production rules, starting from the start symbol $S$. $\epsilon$ is the conventional null terminal symbol. The set of trees generated by $G$ is $L(G)$.

**Example 2.1** Let $G_1 = (\Sigma, N, P, S)$ such that $\Sigma = \{a, b\}$, $N = \{S, X\}$, and $P = \{S \rightarrow a(aSa), S \rightarrow a(bXb), X \rightarrow b(bXb), X \rightarrow b\}$. A sample derivation is given in Figure 1. $L(G_1)$ is thus the set of ternary-branching trees over the symbols $a$ and $b$, where all nodes to a certain depth $D$ are $a$ nodes, and all below are $b$ nodes.

To relate this to a query language, we now review the approach presented in Wood (2003) to relate these regular tree grammars to

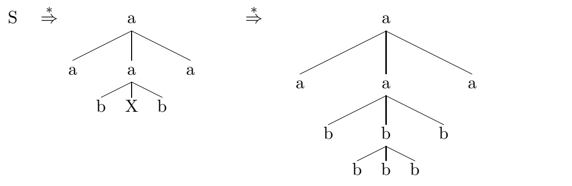

Figure 1: Derivation for RTG $G_1$

XPath (XPath, 1999). XPath is a language for selecting nodes from XML document trees, and is thus an important part of XSLT and XQuery. Expressions in XPath in themselves can be seen as simple queries over trees.

An XPath expression is a mapping from a node (the *context node*) to the set of all nodes reachable by the specified path. A path expression is written as a series of steps where each step defines the *axis* used to reach new nodes and a *node test* used to restrict the set of nodes reached along the axis. Axes include *child*, *descendent*, *following*, *attribute* and *self*. Node tests consist of two parts: a restriction on the element name and an optional predicate expression. Other features, such as built-in functions, are also allowed. Notationally, a null axis stands for the child axis, // the descendent axis, the wild card * any node label, and [] a predicate expression. The full XPath expression definition is fairly complex, and can be found at XPath (1999); here we give an example.

**Example 2.2** [From Wood (2003)] The XPath query $a//b[*/i]/g$ selects nodes labelled with $g$ (called $g$-nodes for short) that are children of $b$-nodes, such that the $b$-nodes are both descendants of the root $a$ node and have an $i$-node as a grandchild. In the left-hand tree in Figure 2, these would be the nodes in bold font. (The return value of the XPath expression would be the node $g$, but here we are only concerned with the trees that would be matched.)

The possibility of arbitary functions for node tests means that XPath can be augmented to be arbitrarily powerful. To investigate questions of formal power, then, only subsets of XPath are examined. Wood's paper notes that, for this reason, a number of other researchers have been interested in the properties of different fragments of XPath, denoted $\mathrm{XP}^{\{[]\}}$, $\mathrm{XP}^{\{[],*\}}$ and $\mathrm{XP}^{\{[],*,//\}}$, depending on which XPath constructs are included in the fragment. The work of Wood himself is on the fragment $\mathrm{XP}^{\{[],*,//\}}$ and the question of whether the containment problem—whether one query is subsumed by another—under a Document Type Definition can be decided in the complexity class PTIME. To demonstrate that this is the case he conceives of XPath queries as *tree patterns* (Abiteboul, 1997; Deutsch et al., 1999), which can be described by regular tree grammars. For example, the XPath query of Example 2.2 could be pictured as the tree pattern on the right in Figure 2, where the dotted line indicates non-immediate dominance.

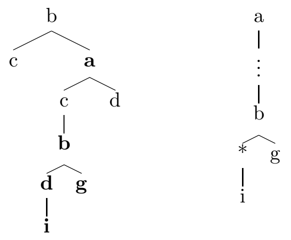

Figure 2: Tree matching $a//b[*/i]/g$, and corresponding tree pattern

An RTG then describes all trees matching this tree pattern. In order to define this RTG, Wood defines some shorthand notation, which we will also adopt. For an alphabet $\Sigma = \{a_1, \ldots, a_k\}$, we write $n \to \Sigma(r)$ for the set of productions $\{n \to a_1(r), \ldots, n \to a_k(r)\}$.

In order to generate an arbitrary tree over $\Sigma$, we define a nonterminal $n_\Sigma$ by the shorthand production $n_\Sigma \to \Sigma((n_\Sigma)^*)$.

**Example 2.3** For the query $Q = a/b$ over alphabet $\Sigma$, the productions for the corresponding RTG are

$$\begin{aligned} n_a &\to a((n_\Sigma)^*\, n_b\, (n_\Sigma)^*) \\ n_b &\to b((n_\Sigma)^*) \end{aligned}$$

An additional shorthand is to compress the ordering of siblings implicit in RTGs. Each permutation of children at a node would require a separate production, so as shorthand the & symbol is used: $a\&b$ represents $ab$ and $ba$ $(a, b \in \Sigma)$.

**Example 2.4** The productions from the query $a[b][c]$ are

$$\begin{aligned}&n_a \rightarrow a((n_\Sigma)^* \,\&\, n_b \,\&\, (n_\Sigma)^* \,\&\, n_c \,\&\, (n_\Sigma)^*)\\ &n_b \rightarrow b((n_\Sigma)^*)\\ &n_c \rightarrow c((n_\Sigma)^*)\end{aligned}$$

**Example 2.5** The productions for the query $a//b$ are

$$\begin{aligned}&n_a \rightarrow a((n_\Sigma)^* \, n_b \, (n_\Sigma)^*)\\ &n_b \rightarrow b((n_\Sigma)^*)\\ &n_b \rightarrow \Sigma((n_\Sigma)^* \, n_b \, (n_\Sigma)^*)\end{aligned}$$

Wood then defines the following procedure for constructing an RTG $G$ from a query $Q$, given an alphabet $\Sigma = \{a_1, \ldots, a_k, *\}$ and $Q$ in $\mathrm{XP}^{\{[],*,//\}}$. First, each node in Q is numbered uniquely, with the root node numbered 1. Then $G$ is given by $(\Sigma, N, P, n_1)$, where $N = \{n_1, \ldots, n_m, n_\Sigma\}$, and $P$ is constructed inductively as follows.

1. If node $i$ in $Q$ is a leaf, then $P$ includes $n_i \rightarrow a_j((n_\Sigma)^*)$ if $i$ has label $a_j \in \Sigma$, or $n_i \rightarrow \Sigma((n_\Sigma)^*)$ if $i$ has label *.
2. If node $i$ in $Q$ has child nodes $j_1, \ldots, j_m$, then $P$ includes $n_i \rightarrow a_l((n_\Sigma)^* \,\&\, n_{j_1} \,\&\, (n_\Sigma)^* \,\&\, \ldots \,\&\, (n_\Sigma)^* \,\&\, n_{j_m} \,\&\, (n_\Sigma)^*)$ if $i$ has label $a_l \in \Sigma$, or $n_i \rightarrow \Sigma((n_\Sigma)^* \,\&\, n_{j_1} \,\&\, (n_\Sigma)^* \,\&\, \ldots \,\&\, (n_\Sigma)^* \,\&\, n_{j_m} \,\&\, (n_\Sigma)^*)$ if $i$ has label *.
3. If node $i$ in $Q$ is connected to its parent by a descendent edge, then $P$ includes $n_i \rightarrow \Sigma((n_\Sigma)^* \, n_i \, (n_\Sigma)^*)$

## 3 XPath Extensions and RTGs

Cassidy (2003) presented some extensions to XPath, based on the requirements of typical linguistic queries. An example is that of finding matches for a given syllable structure: "Find sequences of any number of consonant phonemes followed by a single vowel following by any number of consonants". This would require a regular expression (C+VC+) over the `/following` axis. Under the current definition of XPath, it is not possible to specify regular expressions over axes; nor is it possible to specify more complex definitions of where conditions on nodes should be applied. Cassidy consequently specifies an extension that allows this, which has the following components:

**axis** This is as for the axes in the standard XPath definition.

**step** This determines how many steps should be taken along the axis, and takes the form of a list of positive integers or a special value `inf`. A path length is allowed if it matches one of the integers in the list; any path length is allowed if only the value `inf` is given; path lengths greater than the highest integer are allowed if the list contains both integers and the value `inf`. For example, $[2, 3, 4]$ allows paths only of lengths 2, 3 or 4; $[1, 10,$ `inf`$]$ allows paths of length 1 or of length greater than 10.

**condition** This is a general boolean condition on nodes in the same sense as XPath.

**where** This specifies where on the path the condition must hold, via a list of positive integers or the special values `inf` or `end`. An integer in the list means that the condition is applied to that node; `inf` means that the condition must hold for all nodes in the path; `end` means that the condition applies to the final node on the path. For example, the list $[1, 2,$ `end`$]$ would find paths where the condition was satisfied by the first, second and last nodes on the path.

A proposed syntax for this, in keeping with the XPath syntax, would be

```
/axis{step}::condition::{where}
```

The **step** and **where** components would default to their XPath values (1 and `end` respectively).

**Example 3.1** Cassidy gives an example for the C+VC+ query which uses the `/following` axis. An analogous example using the axes already presented in Section 2 would be one to match trees with a chain of VP nodes followed immediately below by a chain of NP nodes, such as the one in Figure 3. Such a query could be expressed as

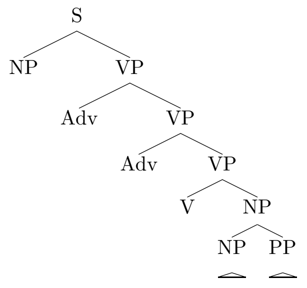

Figure 3: Tree with sequence of VPs and NPs

```
/child{0,inf}::VP::{0,all}
/child{0,inf}::NP::{0,all}
```

The question is then, Can this extension be represented by RTGs? The intuition is that it can—the essence of this aspect of the extension is to allow regular expressions over paths, which is also the essence of RTGs. We demonstrate this below by giving a construction of an RTG for each extended XPath query. As with the construction in Section 2, we will look at just a restricted subset of this XPath extension: **axis** `/child` and `/descendent` (which is in fact just an infinite `/child`); **step** as defined above; **condition** only labels on nodes, or *; and **where** as above. We call this XP-ext.

**Example 3.2** The query in Example 3.1 would be represented by the RTG $(\{VP, NP, *\}, \{n_S, n_V, n_N\}, P, n_S\})$, where $P$ is the set of productions

$$\begin{aligned} n_S &\to \Sigma((n_\Sigma)^* \, n_V \, (n_\Sigma)^*) \\ n_V &\to VP((n_\Sigma)^* \, n_V \, (n_\Sigma)^*) \\ n_V &\to VP((n_\Sigma)^* \, n_N \, (n_\Sigma)^*) \\ n_N &\to NP((n_\Sigma)^*) \\ n_N &\to NP((n_\Sigma)^* \, n_N \, (n_\Sigma)^*) \end{aligned}$$

To derive an RTG corresponding to a query in XP-ext, we identify the corresponding tree pattern(s). First, to allow regular expressions over paths in tree patterns, we make a further notational extension, either adding the symbol $c$ next to the non-immediate dominance link in the tree pattern, to indicate that the condition holds over the non-immediate dominance link, or the symbol $*$ if it does not.

Now, we need to look at two separate cases. The first case is the general one: in aiming only to match sets of trees by tree patterns, only one tree pattern is generally necessary, regardless of the number of elements specified in the `step` and `where` components. This is because although one pattern tree might be expected for each `step` value, in fact one describing the smallest step suffices, as it subsumes any other. For example, in the query `/child{2,4}::S::{2,4}`, any tree which matches the appropriate pattern tree of height 4 will match the pattern tree of height 2. Note that these different **step**s and **where**s are differentiated with respect to the XPath return values; however, that is beyond the scope of this paper, which concerns itself only with trees matched.

The second case deals with the specific case of `where` value `end`; the subsumption relationship does not hold here. For example, consider the query `/child{2,4}::S::{end}` over trees consisting of a single path. There will be trees where the end of the path (`step`) of length 2 (matching the label S) do not have an S node at the end of the path of length 4, and vice versa; that is, there is no subset relation between the sets of trees specified for `step`s of lengths 2 and 4.

We define these two cases below. In these definitions, we take **axis** `/child` and **condition** (i.e. node label) `c`.

**Case 1** For query $Q$ with **step** `[`$i_1, \ldots, i_n,$`inf]` $(i_1 < \ldots < i_n)$, and **where** `[`$j_1, \ldots, j_m,$`inf]` $(j_1 < \ldots < j_m <$ `inf`$)$, we construct a tree pattern of height $i_p$, where $i_p$ is the smallest element of $[i_1, \ldots, i_n,$`inf`$]$ with label `c` for each node $j_p$ $(p < m, j_p < i_p)$. If $i_p$ is `inf`, we label the non-immediate dominance edge `c` if **where** contains `inf`, * otherwise.

**Case 2** For query $Q$ with **step** `[`$i_1, \ldots, i_n,$`inf]` $(i_1 < \ldots < i_n)$, and **where** `[end]`, we construct one tree pattern for each element of **step**, with the last node in each tree pattern labelled `c`, and any non-immediate dominance edge labelled $*$.

**Example 3.3** The query in XP-ext

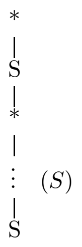

Figure 4: Pattern tree for Example 3.3

`/child{inf}::S::{1,3,inf}` would give the pattern tree in Figure 4.

Our construction for an RTG is then as for the construction in Section 2, but with part 3 replaced by with 3':

3' If node $i$ in $Q$ is connected to its parent by a non-immediate dominance edge labelled with $a \in \Sigma$, then $P$ includes $n_i \rightarrow a((n_\Sigma)^* n_i (n_\Sigma)^*)$ or if unlabelled, then $P$ includes $n_i \rightarrow \Sigma((n_\Sigma)^* n_i (n_\Sigma)^*)$

## 4 Non-Regular Queries

However, some queries that are of interest to (computational) linguists are not able to be expressed by RTGs. There was much discussion in the mid- to late-twentieth century regarding whether natural languages could be described by context-free (string) grammars (CFGs). A fairly common belief was that they could not be, accompanied by attempts to prove this; Pullum and Gazdar (1982) refuted these earlier arguments, and it was not definitively shown that natural language as a string language was not context-free until the work of Shieber on Swiss German (Shieber, 1985).

Bresnan et al. (1982), in addition, made an argument based on syntactic structure, using the example of Dutch. We represent their argument, where they show that an RTG cannot describe this kind of structure—which obviously has some linguistic interest—and hence neither can any tree query language based on it. We then consider what formal tree machine is minimally needed for describing natural language.

Cross-serial dependencies occur when dependencies in a sentence are interleaved with each other, such that in a string $a_1 a_2 \ldots a_n b_1 b_2 \ldots b_n$ there are dependencies between elements $a_i$ and $b_i$ ($i \in \{1 \ldots n\}$). A well known example from Dutch, given in Bresnan et al. (1982), is in Figure 5.

Pullum and Gazdar (1982) showed that it was possible to describe the string language using a CFG, but noted that the associated structure would not necessarily be useful. Bresnan et al. (1982) then comprehensively investigated what structures would be appropriate on linguistic grounds. One proposal they considered was for a flat structure of NPs, with right-branching VPs, as in the leftmost tree of Figure 5, with evidence for the right-branching VPs coming from possibilities of conjunctions in Dutch. However, they noted that the sequence of NPs has more constituent structure than indicated in the leftmost tree of Figure 5, and conclude that the structure in the centre of Figure 5 is the one that is consistent with the data. They note that this proposed structure should be uncontroversial, as it embodies only predicate-argument relations, rather than any aspects of syntax whose representations may be more open to question.

They use a pumping lemma for regular tree languages to demonstrate that sets of these sorts of trees from Figures 5 cannot be generated by RTGs. Comon et al. (1997) give a more precise definition, but broadly, for any tree $T$ of height greater than some constant $k$ in a tree language $L$, there is a non-trivial segment of $T$ that can be 'pumped' in a manner analogous to in the pumping lemma for regular string grammars; and any tree formed from $T$ with an arbitrary number of these segments inserted appropriately will also be in $L$. In the case of the centre tree of Figure 5, there is no segment of the tree that can be chosen to be pumped that will maintain equal numbers of NPs and Vs (that is, specifying the same tree language). Roughly speaking, there cannot be counting or matching of numbers of internal symbols in tree patterns.

We note here that there is an additional possibility that they do not discuss that is also consistent with the data, differing only in the placement of the subtree headed with the $V'$; this is the rightmost tree in Figure 5. The same pumping lemma shows that the set of these trees also cannot be generated by an RTG.

## 5 Context-Free Tree Grammar

What, then, can generate these sets of trees? One possibility is a context-free tree grammar (CFTG). The basic idea of these is that, whereas trees generated by RTGs have regular paths, CTFGs have context-free paths.

CFTGs were introduced in Rounds (1970). We do not have space for a full formal presentation of them, and we would also note that unlike the RTGs defined above they are defined over ranked trees. However, here is a brief definition, along with an example. A *context-free tree grammar* $G$ is a 4-tuple $(F, \Phi, P, \mathcal{K}_0)$ where $F$ is a finite ranked alphabet; $\Phi = \{\mathcal{K}_0, \mathcal{K}_1, \ldots, \mathcal{K}_n\}$ is a finite ranked alphabet of nonterminals; $P$ is the set of production rules, a finite set of pairs $(\mathcal{K}_i(x_1, \ldots, x_m), t_x)$, where $i = 0, \ldots, n$, $K_i \in \mathcal{F}$, $x_i$ are variables, and $t_x$ is a tree over $F$, $\Phi$ and variables $x_i$; and $\mathcal{K}_0$ is the initial nonterminal.

An application of a production rule to a tree $T$ involves choosing a nonterminal $\mathcal{K}_i$ with $m$ children, taking a rule with lefthand side $\mathcal{K}_i(x_1, \ldots, x_m)$, identifying the $m$ subtrees of $\mathcal{K}_i$ in $T$ with the variables $x_1, \ldots, x_m$, and replacing the chosen subtree of $T$ rooted in $\mathcal{K}_i$ with the righthand side of the rule along with appropriately substituted variables $x_1, \ldots, x_m$.

No CFTG is possible for the leftmost tree of Figure 5, as symbols are ranked, and that tree would require an infinite number of ranked symbols with label *S* to describe the unbounded number of children *NP*.

A CFTG to describe the centre tree of Figure 5 would be $G = (F, \Phi, P, \mathcal{K}_0)$, where $F = \{S, NP, VP, V'\}$, $\Phi = \{\mathcal{K}_0, \mathcal{K}_1\}$, and $P$ is the set of productions given in Figure 6.

A CFTG to describe the rightmost tree of Figure 5 would be $G' = (F, \Phi, P', \mathcal{K}_0)$, where

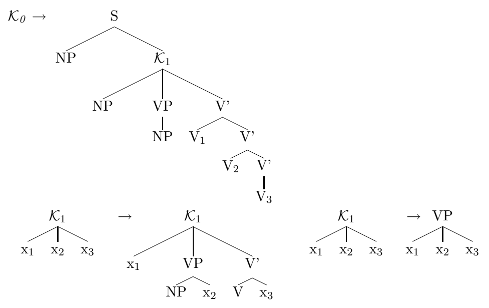

Figure 6: CFTG for cross-serial option 2

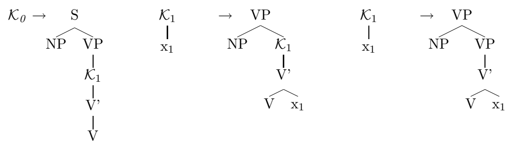

Figure 7: CFTG for cross-serial option 3

$F$ and $\Phi$ are as for $G$ above, and $P'$ is the set of productions in Figure 7.

In essence, CFTGs allow 'counting' throughout a tree—in the examples above, the counts of NPs and Vs match—in the same way as CFGs allow counting in a string. Thus CFTGs might be a suitable backbone for a tree query language. However, they are not computationally very attractive. Just as the string languages yielded by RTGs are the context-free languages, the string languages yielded by CFTGs are the indexed languages, which include exponentially increasing languages such as $\{a^{2^n} \mid n \geq 1\}$ that are not a feature of any human language.

However, the sets of productions $P$ and $P'$ are actually quite different. It is straightforward to demonstrate that those in $P'$ are in fact *spinal-formed* under the definition of Fujiyoshi and Kasai (2000). In essence, a spinal-formed CFTG disallows duplication of counts along different paths (as in the duplicate counts of NPs and Vs in separate subtrees of the centre tree of Figure 5). Spinal-formed CFTGs form a proper subset of CFTGs with much restricted power; interestingly, their string languages have been proved by Fujiyoshi and Kasai (2000) to be the class of mildly context-sensitive languages, and recent work (Fujiyoshi and Kawaharada, 2005) has included promising

(1)

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| …dat | Jan | Piet | Marie | de kinderen | zag | helpen | laten | zwemmen |
| …that | Jan | Piet | Marie | the children | see-PAST | help-INF | make-INF | swim-INF |

…that Jan saw Piet help Marie make the children swim

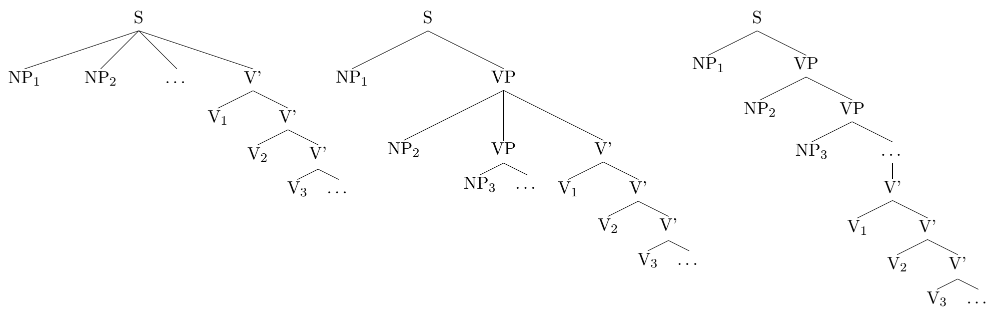

Figure 5: Proposals for cross-serial dependencies

results on recognition complexity.

This rightmost tree of Figure 5, then, establishes that a language to query trees representing the syntax of natural language requires an underlying tree machine beyond RTGs, but not necessarily beyond the power of spinal-formed CFTGs.

An additional result of Fujiyoshi and Kasai (2000) is the definition of a Linear Pushdown Tree Automaton (L-PDTA) that recognises exactly the class of trees generated by spinal-formed CFTGs. These are similar to the automata that recognise the tree sets of RTGs, but have a non-duplicable stack in operation as the automaton walks a path of a tree; that is, the stack can only be passed along a single branch of the tree.

This suggests that a similar mechanism might be appropriate for a tree query language that would allow limited counting. Using the notation of Section 3, we might have a query to find trees with (not necessarily balanced) NPs and Vs with the structure of the rightmost tree of Figure 5) as follows:

```
/child{0,inf}::VP[/child::NP]::{0,all}
/child::VP
/child{0,inf}::V[/child::V']::{0,all}
```

We would then extend this so that the number of nodes matching `/child{0,inf}::VP[/child::NP]::{0,all}` would be one less than the number matched by `/child{0,inf}::V[/child::V']::{0,all}` in order to match only those trees with appropriately paired NPs and Vs.[1] Notationally, this might be represented as:

```
/child{X=|{0,inf}|}::VP[/child::NP]::{0,all}
/child::VP
/child{X=|{0,inf}|}::V[/child::V']::{0,all}
```

where $X$ is a variable containing the count of nodes matched by the particular components of the query, and $|Y|$ represents the number of steps actually matched for list of steps $Y$.

To restrict this to match only those trees describable by spinal-formed CFTGs, passing counts down through predicate expressions, which would in effect permit stack duplication, is disallowed (e.g. `VP[/child::NP]`).

However, this is just an indication of the form that an XPath-like query language based on a spinal-formed CFTG might take, and is intended only to be the starting point for future work.

## 6 Conclusion

The aim of this paper has been to examine what formal machinery is necessary for linguistic tree query languages. Existing tree query languages are typically related to regular tree grammars, although these query languages are almost exclusively for non-linguistic XML documents. The first contribution of the paper has been to show

[1]The topmost NP in the tree, under the S, is not matched by the query.

that regular tree grammars can be used as the basis for a range of proposed extensions to XPath motivated by linguistic considerations, for very typical sorts of queries such as those representing a search for a regular expression over axes. The paper's second contribution has been to demonstrate that regular tree grammars cannot, however, be a basis for some queries of linguistic interest. We have shown that machines with at least the power of spinal-formed context-free tree grammars, with their limited ability to count, can describe those constructions that are beyond RTGs, and made initial suggestions on how this ability to count could be incorporated into a query language.

There is much scope for future work in this direction. To deal with XPath return values we are interested in transducers based on (subtypes of) context-free tree grammars which have not yet been defined; many of the properties of the formal mechanisms remain to be investigated; and, most relevant to this paper, it is an open question as to how precisely a query language can be defined based on this mechanism.

## References

Serge Abiteboul. 1997. Querying Semi-Structured Data. In Foto Afrati and Phokion Kolaitis, editors, *Database Theory—ICDT'97*, pages 1–18. Springer-Verlag.

Joan Bresnan, Ronald Kaplan, Stanley Peters, and Annie Zaenen. 1982. Cross-serial Dependencies in Dutch. *Linguistic Inquiry*, 13(4).

Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. 2001. Regular tree and regular hedge languages over unranked alphabets. Technical Report HKUST-TCSC-2001-05, Hong Kong University of Science and Technology.

Steve Cassidy and Jonathon Harrington. 2001. Multi-level annotation in the Emu speech database management system. *Speech Communication*, 33(1–2):61–77.

Steve Cassidy. 2003. Generalizing XPath for directed graphs. In *Proceedings of Extreme Markup Languages 2003*.

Boris Chidlovskii. 2000. Using Regular Tree Automata as XML schemas. In *Proceedings of IEEE Advances in Digital Libraries*, pages 89–104.

H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. 1997. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`. Release October 1st 2002.

Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. 1999. A Query Language for XML. In *Proceedings of the 8th International World Wide Web Conference*, pages 77–91.

A. Fujiyoshi and T. Kasai. 2000. Spinal-Formed Context-Free Tree Grammars. *Theory of Computing Systems*, 33:59–83.

A. Fujiyoshi and I. Kawaharada. 2005. Deterministic Recognition of Trees Accepted by a Linear Pushdown Tree Automaton. In *Proceedings of the Tenth International Conference on Implementation and Application of Automata*.

Esther König and Wolfgang Lezius. 2001. The TIGER language—a description language for syntax graphs. Part 1: User's guidelines. Technical report, IMS, University of Stuttgart.

Catherine Lai and Steven Bird. 2004. Querying and Updating Treebanks: A Critical Survey and Requirements Analysis. In *Proceedings of the Australasian Language Technology Workshop 2004*.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Makoto Murata, Dongwon Lee, and Murali Mani. 2000. Taxonomy of XML Schema Languages using Formal Language Theory. In *Proceedings of Extreme Markup Languages 2000*.

Geoffrey Pullum and Gerald Gazdar. 1982. Natural languages and context-free languages. *Linguistics and Philosophy*, 4:471–504.

Douglas Rohde, 2001. *TGrep2 User Manual*.

William Rounds. 1970. Mappings and Grammars on Trees. *Mathematical Systems Theory*, 4:257–287.

Stuart Shieber. 1985. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343.

Peter Wood. 2003. Containment for XPath Fragments under DTD Constraints. In *Proceedings of the 9th International Conference on Database Theory*, pages 300–314.

XPath. 1999. XML Path Language (XPath), Version 1.0. Available on: `http://www.iw3.org/TR/xpath`.